Computers in Industry 128 (2021) 103433

Contents lists available at ScienceDirect

Computers in Industry

journal homepage: www.elsevier.com/locate/compind

Monte Carlo Tree Search for online decision making in smart industrial production

Richard Senington*, Bernard Schmidt, Anna Syberfeldt

School of Engineering Science, University of Skövde, Sweden

ARTICLE INFO

Article history:
Received 31 August 2020
Received in revised form 22 February 2021
Accepted 23 February 2021
Available online 16 March 2021

Keywords:
Monte Carlo Tree Search
MCTS
Human-robot collaboration
HRC
Discrete event simulation
Dynamic scheduling

ABSTRACT

This paper considers the issue of rapid automated decision making in changing factory environments, situations including human-robot collaboration, mass customisation and the need to rapidly adapt activities to new conditions. The approach taken is to adapt the Monte Carlo Tree Search (MCTS) algorithm to provide online choices for the possible actions of machines and workers, interleaving them dynamically in response to the changing conditions of the production process. This paper describes how the MCTS algorithm has been adapted for use in production environments and then the proposed method is illustrated by two examples of the system in use, one simulated and one in a physical test cell.

1. Introduction

In traditional mass production systems, the focus has been on the production of identical products with high throughput. To produce as much as possible it has been important to focus on optimising the flow of components through the factory. A variety of methods have been utilised for this including linear programming, genetic algorithms and discrete event simulation. Mass production has also been supported by automation technology, either in the form of specialised machines or modern general-purpose robots. This traditional approach has made use of *static* automation and optimisation, where the optimisation tools provide a plan of tasks to be repeated with limited variation and the machines are then statically coded to operate the plan. This locks machines and cells to limited ranges of tasks and requires substantial time both to develop new lines and adapt old lines to new products.

Traditional mass production can be contrasted with the current trend in manufacturing towards mass customisation and individualisation of products for the users (Sandrin et al., 2014). This is sometimes called lot-size-one, a name that illustrates the most extreme situation where every product in the factory is unique. The problem becomes how to build a factory that can support such variation in product requirements and what machines does such a factory need. There are a variety of approaches to this problem being investigated including modular production (Telgen, 2017), reconfiguration (Koren, 2010; Koren et al., 2018) and human-robot collaboration (HRC). In all these cases it is necessary to plan sequences of tasks and operations to maximise the productivity of the system, to do so for very varied product requirements and frequently on short notice.

This places the problem, and this paper, within the field of dynamic scheduling, however (Ouelhadj and Petrovic, 2008) points out that theoretical work in scheduling often provides techniques for *static* scheduling rather than techniques for planning and reacting in the real world. The field of dynamic scheduling has been broken down along a range of lines including when scheduling should take place (periodically or in response to events), the degree of schedule planning (dispatch rule methods being very short term, predictive-reactive maintain a schedule to be modified and pro-active methods attempt to provide schedules that will fulfil performance requirements despite the dynamic environment) and the type of algorithms used (for example metaheuristic, agent-based or classical search) (Ouelhadj and Petrovic, 2008). More recently machine learning has been employed within the field to improve the predictive capabilities of dynamic scheduling (Morariu et al., 2020) and multi-objective optimisation has been employed to try to manage the trade-offs between competing requirements of industry such as energy usage against raw productivity (Tang et al., 2016).

* Corresponding author at: Högskolan i Skövde, Högskolevägen Box 408, 541 28, Skövde, Sweden.
E-mail addresses: Richard.james.senington@his.se (R. Senington), bernard.schmidt@his.se (B. Schmidt), anna.syberfeldt@his.se (A. Syberfeldt).

https://doi.org/10.1016/j.compind.2021.103433

Within HRC the use of dynamic scheduling is similarly limited in practice and as noted in (Krüger et al., 2017) few studies work with advanced dynamic scheduling. Some examples of research where dynamic planning has been used include (Valente et al., 2012) which proposed a two-level architecture for responding to errors in the production process. This system consists of a top-level planner which creates a plan to be executed in the same way as a static system. A diagnostic component then detects if an error occurs at runtime and requests that the top-level scheduler performs an additional execution to repair the primary plan. Simultaneously, the system uses the second level of planning for dynamic recovery while waiting for the new primary plan. Another approach from (Nikolakis et al., 2018) involved a hierarchical model of the tasks for planning where plans are initially generated and then replanned in the event of an issue. Related research is also being conducted on the use of HRC for non-industrial applications and one key result has been the use of Partially Observable Markov Decision Process (POMDP) to model expected behaviour in a human partner for a robot, enabling collaboration in changing a diaper on a doll (Brown and Tellex, 2015). A final example comes from (Dahl et al., 2017) who make use of a model of a production cell using an Extended Finite Automaton which includes initialisation and completion events with guard conditions to control transitions.

This paper explores an alternative method for making automatic decisions in response to dynamic conditions, with a focus on operations within single production cells. The foundation of the approach is the Monte Carlo Tree Search (MCTS) algorithm (Abramson, 1987; Bouzy and Helmstetter, 2004; Coulom, 2006; Chaslot et al., 2008a, b) a relatively new technique that has had considerable success in the field of games such as Go (Silver et al., 2016). MCTS has been successful because it has been shown to limit the combinatorial explosion of search spaces in problems such as Go, but it can be applied more generally to problems that can be modelled as trees of choices (Browne et al., 2012). A variety of specialisations for the MCTS approach have been explored in connection to a range of possible problems, as can be seen in this survey of the field (Browne et al., 2012).

While the main focus of research within MCTS has been competitive games, variations and research has been conducted that focus on single-player games (Liebana et al., 2012; Samothrakis et al., 2014) and cooperative games (Williams et al., 2015). This direction of research suggests that MCTS could also be appropriate for factory management, seeing the task of running a factory as either the cooperation of agents towards a common goal or as a group of operators being managed by a singular intelligence. MCTS has also been previously used in HRC (Toussaint et al., 2016), where they adapt MCTS to include simultaneous operations acting in real-time and apply the result to an HRC laboratory demonstration. The capability of the algorithm family to perform strongly on problems without domain-specific knowledge would also reduce the required effort to deploy this type of solution.

To the authors' knowledge, this algorithm family has not been used for dynamic production control apart from the previously mentioned HRC example, however other variants of the MCTS family of algorithms have been tested for problems in planning and robotics. In the context of robot motion planning for a single robot (Kartal et al., 2016) utilised MCTS for path planning in a range of benchmark problems and compared the quality of results to known optimums. MCTS has been examined for logistics (Trunda and Barták, 2013; Edelkamp et al., 2016) and production planning (Chaslot et al., 2006; Lubosch et al., 2018), fields directly connected to industrial application. Research is also ongoing in extending MCTS to wider ranges of problems and this has resulted in Monte Carlo Action Programming (MCAP) (Belzner, 2017), where an environment is modelled as states and actions transitioning between sets of states, searched over by MCTS. MCAP has been applied to at least single robot problems and provides a programming model that does not require the programmer to concern themselves with the details of the sequencing of operations.

Our contribution is to combine MCAP with the real-time MCTS approach of (Toussaint et al., 2016) to ease configuration of a dynamic control system. The completed concept is demonstrated on two example cases, both modelled after industrial assembly applications, utilising HRC. A conversion method is provided to transform a model of a cell from the actions and state variables of action programming to the "initiation and termination of activities" and their concurrent execution found in (Toussaint et al., 2016). A further modification to the search process is presented to search for combinations of possible initiation actions after each possible termination signal, a modification elaborated on in section 2.

2. Adapting MCTS

This section first reviews the standard implementation of MCTS, drawn from (Chaslot et al., 2008b). It then describes our implementation and the changes that were made to support the real-time execution of decision making.

2.1. Monte Carlo Tree Search

MCTS can be viewed as a general-purpose heuristic which can be applied to many problems, in that it assigns weights to nodes of the tree and when stopped selects the best node as the choice. Where it differs from traditional heuristics is that it builds the weight of each node by repeatedly exploring the tree below the node randomly until the search completes, a process known as the simulation phase. The weight of a node is then the average of the results of these simulations. The intuition is that if the simulations beyond a particular choice tend to result in an overall bad result, this is a poor choice. Consider Chess, imagine there are two possible moves before the player and for each move the random search of the tree is run several hundred times. When looking at the weightings, one choice resulted in an overall loss 300 times in 400, while the other was a loss 100 times in 400. The player would be expected to choose the result where they tended to win.

The algorithm combines the process of random simulations with a gradual expansion of the tree from the root. It does this by expanding the tree by one node at each iteration and generating a single random simulation from that new node. The results of the simulation are then propagated back up the spine of the tree to the root, updating the weights of the nodes. This leads to an iterative cycle of selecting a leaf to expand, expanding that leaf, simulating beyond the leaf and updating the weights. This cycle can be seen in Fig. 1, which also illustrates the process of a gradually increasing tree over several iterations. The cycle can be interrupted at any stage and the option that has the best value at that point selected, though as with traditional methods the longer it runs for the more complete the search and the better the estimation of the quality of the options you would expect to get.

2.2. Events, states & search

The standard MCTS algorithm's primary application area has been in games where after some number of turns, or actions, the game will end and a player will win. This provides a direct measure of the quality of a choice, does it lead to victory or defeat. In the case of the production industry however there may be no clear end state, only a desire to continue producing parts for the foreseeable future, and with as high throughput as possible. Production industries are also not competitive games in the same sense as a game like chess. They are either collaborative games or single-player games. We model the search process as a single-player game where the

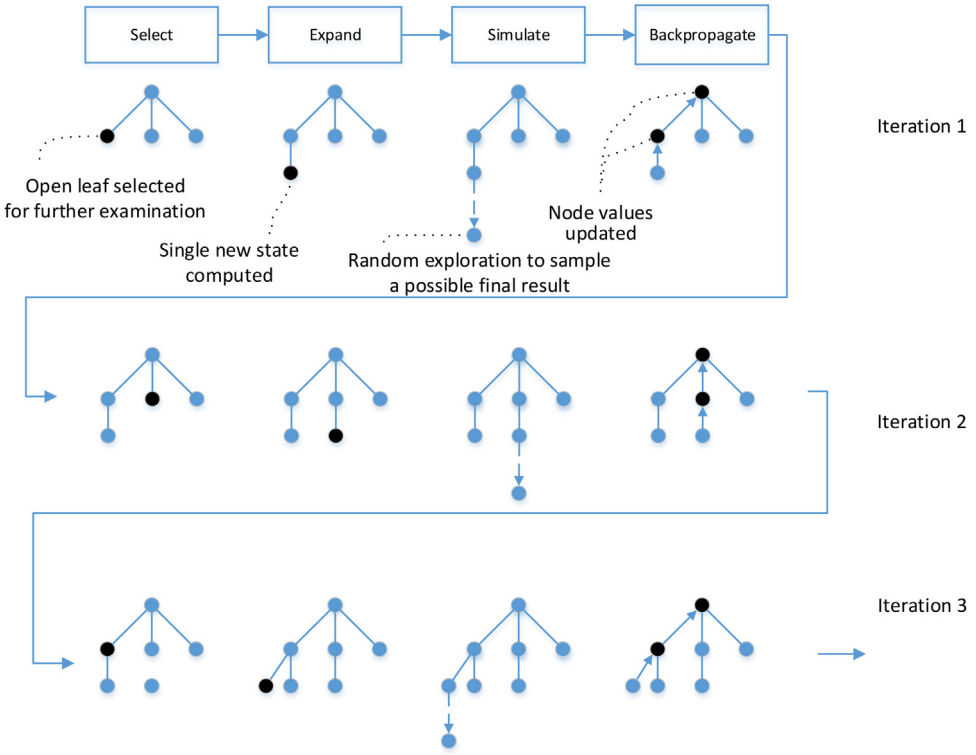

Fig. 1. The evolution of an MCTS search process. Adapted from (Williams et al., 2015). Note how in each iteration the tree grows but also the weighting of the nodes updates and evolves as the number of simulations increases.

player is the cell coordinator, managing the various tools, robots and workers.

We follow the approach in (Toussaint et al., 2016) of using the terms state and action for the model of the decision process where actions connect states. We then use the term *activity* for running concurrent operations within the cell. A production system is modelled using a set of discrete variables. Each variable represents the current status of some aspect of the production process and more usually some physical entity in the cell such as a robot gripper. The set of possible values each variable may take is created while modelling a process or environment and in the case of the gripper might include; holding, not-holding, taking and releasing. The values for taking and releasing would be used when an activity is being performed and the values holding and not-holding would be used when activities complete. A node in the search tree consists of;

- a set of variable assignments,
- the current time or expected time when that node is reached and
- a list of executing activities as in (Toussaint et al., 2016).

The actions connecting the state nodes of the tree are of 2 types. The first are *begin* actions, which begin a single activity, adding the activity to the list of running activities and updating some or all of the variables. Begin actions do not add any time to the current time stored in the nodes of the search process. The second are *completion* actions, which indicate that an activity has ended. These remove an entry from the list of running activities, change some or all of the variables and do update the time value with the expected time of the completion of the activity. The expected times of activities are not treated as probability functions but as an expected value to simplify the implementation, this is a limitation that will be returned to in future research. The time update of completion actions is used within the search process and when a *completion* event occurs in a running system the measured time cost of the activity will be used instead.

Due to the potentially unlimited execution time of a production cell and the limit on processing time, the search process and the algorithms simulation phase include a depth limit based on the execution time of the production process. In other words, the simulation cannot proceed beyond a node which has too high a time value, and this time limit is recomputed periodically as events from the real system are recorded. This enforces termination of the search process in a similar manner to previous experiments that have been performed by other researchers (Keller and Helmert, 2013; Samothrakis et al., 2014).

The objective of the system is to maximise productivity which we will compute as the number of jobs completed over time, where a job is a series of activities in that cell before the product moves on. The state model does not directly include an indication of the completion of a job, only the completion of activities. To solve this we add a production progress score to each activity. These are then set by the production designer to indicate which activities are intermediate steps and which mark progress. In the examples found in the next section, we have used 0 for all intermediate activities, such as the robot arm simply moving between locations and 1 for activities that contribute directly to progress in an assembly, such as attaching a component. When a simulation ends due to the time limit applied to the search process we sum the progress scores of the completed activities that have been utilised. Hence the value of a given node in the search tree becomes;

$$\frac{1}{|S|} \sum_{\substack{1 < i < |S| \\ 1 < j < |S_i|}} S_{ij}$$

Where S is the set of simulated activity sequences, S_i is the i^{th} simulation and S_{ij} is the progress value of the j^{th} activity of the i^{th} simulation.

2.3. Expansion and selection

As previously described in section 2.1, the standard implementation of MCTS includes an *expansion* and a *selection* phase in the process. These will select a leaf of the currently constructed tree to explore from, add a new child node to the tree and perform a simulation from that child node. This is well suited to turn-based games such as Go and Chess, however, our application includes the following additional issues;

1 At any given time there can be several activities being performed at once. For example, when the robot and worker are performing different tasks we cannot be sure which will finish first and we require a plan for these different situations.
2 The nodes of the tree include an estimate of time, to allow for the depth limit previously described. When an activity completes this time value is unlikely to be correct and so all nodes would require their time to be updated, a potentially slow process.
3 When any activity completes and the variables are updated several alternatives might be possible at once. For example, if a collaborative activity that required both the worker and robot completes then several activities might become available that the worker could perform next. Similarly, the robot might also be able to begin one of several activities. Ideally, it is these *groups* of activities that could be activated that are scored, rather than individual decisions in the search tree.
4 It is usually the case that once a set of new activities have been begun the system will have a relatively long period for preparing for the next phase, though this will vary depending on the application. Hence it is desirable to begin a group of activities quickly before returning to searching, rather than performing the search process while the manufacturing resources stand idle.

Issue 2 points out that when an activity ends the precise time is recorded and this disrupts the MCTS search process by requiring that all nodes of the tree, which would be preserved, would have to be updated. At this time we have simplified this issue by restarting the process with the current status of the cell each time a signal is received. Improving upon this would allow for greater expansion of the tree between events and allow for increasing the depth limits and so this is also a topic for further research.

To resolve the remaining concerns the expansion process was replaced with an initial exhaustive search of a section of the tree. Specifically, the root node is expanded through the actions that represent the completion of the running activities. Subsequently each of these expanded nodes is expanded exhaustively through actions that begin new activities until no further activities can be started. The search process would then be required to wait until the next activity completed before making the next decision. The intention here is to guarantee that for each completing activity all the possible plans, that is sets of subsequent activities, that could be activated have been considered and evaluated to some extent. This tree is not expanded further until a new completion action is received.

To evaluate the possible options equally, a uniform selection strategy is used over the available leaves of the exhaustively searched tree. This is performed cyclically and where time is available will continue looping over the leaves. A simulation is conducted from each leaf and the scores of the expanded nodes updated as normal. Finally, each path of decisions leading to each leaf is recorded as a group and scored so that the group with the best score is ready for activation on the next completion signal.

These modifications to the MCTS algorithm result in this version behaving rather like a combination of search and Discrete Event Simulation (DES), though a DES that is simulating choices that might be made in the future, not only the probability of events.

2.4. Execution of the search process

The practical implementation used was set up to explore the possible options while the system waited for the next completing activity. Each iteration of the MCTS was required to finish but a check was done on incoming signals between iterations and once a completing activity was seen the best set of subsequent actions were sent back. A flow chart of the application design can be seen in Fig. 2.

Activities that are executed in the system were either control procedures interacting with machines or instructions sent to human operators. Where instructions were needed for human operators the test application made use of a tool for storing and displaying instructions on a range of devices (Kardos et al., 2018).

2.5. Describing a cell

Though the focus of this work has not been the modelling of a cell, the usability of the proposed approach can be improved through the use of a simplified description of operations and when they can be performed. The approach taken is presented here for completeness.

The user describes the cell in terms of variables usually representing physical components of the cell that can be changed as a process is run. The user also describes the possible values each variable can take. Each activity is given as a set of pre-conditions, post-conditions and the operation to be performed as in (Belzner, 2017). Each activity includes the time it is expected to take and linking code needed for communication with the physical systems. The description is simplified when compared to the model that will be used for the runtime so that the user does not define the completing actions of the system directly, nor how many activities can execute at the same time.

Cells are often modelled using a hierarchy of components and the same approach is taken here. A user is expected to create a tree of components where a variable is a leaf of the tree. Each leaf has several values it can take. An example of this approach can be seen in Fig. 3 while an example of an activity can be seen in Fig. 4. Variables can also model stages in a production process, for example; indicating if part A has been attached to part B yet.

The simplified model of possible activities and variables is converted to the concurrent activity model in the following way.

- Each value a variable can take is split into two values, the original and an *unavailable* version. The unavailable version corresponds to variable values which would have been used while a specific activity was in progress. For example, the gripper described in section 2.2 had a holding and *releasing* value. The *releasing* value would correspond to a generated unavailable value and is seen in Fig. 4 as *Holding-*.
- Each activity is split into two actions, a beginning and completing action. The start action connects states, one where the prerequisite values are those that the user gave as the prerequisite of the activity and one where the variables have taken the unavailable version of the value.
- The completing action connects states where the variables have their values set to the unavailable values of the pre-conditions

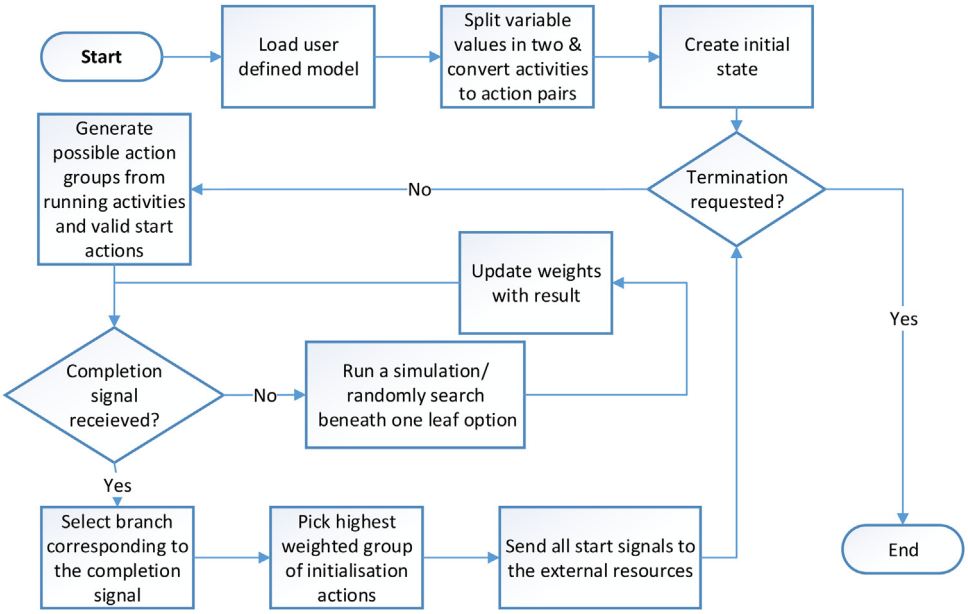

Fig. 2. A flow chart of the high-level execution steps of the system.

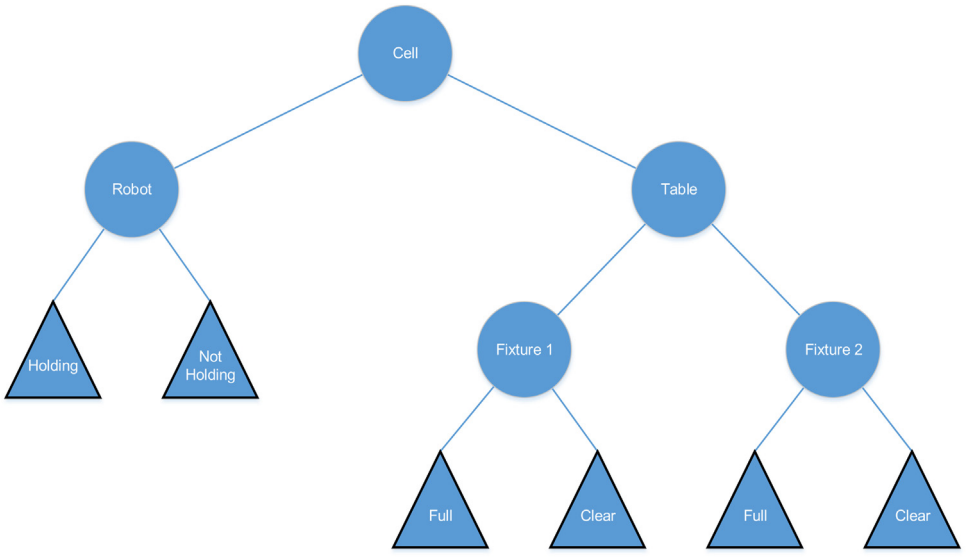

Fig. 3. Modelling a production cell's resources and the states of the resources using a hierarchical tree.

and states where the variable values are those of the original activity post-conditions.

An illustration of a pair of events after transformation can be seen in Fig. 5, which is the transformation of the action seen in Fig. 4.

3. Examples of usage

To illustrate the application of the system two examples are used, one artificial to show how it can choose to interleave tasks while the last is an example from a physical test cell. To depict the decision process in operation, one or more charts are presented of either a simulated or real run, depending on the examples. These charts look like Gantt charts but it should be emphasised that they are reports of activity rather than plans. Each chart represents a single run, either in reality or in simulation, but repeated trials did not introduce any large variations in the execution of these processes suggesting that these charts are a reasonable demonstration of the results of the system. In these examples, the focus of the cell model for actions is the robots and other equipment, and these have the most possible status variations. The human workers have limited status information, restricted to available or not available.

3.1. Varying team sizes

This example shows a case where the size of the team is not fixed. The model includes; a robot arm, either one or two workers, an AGV that carries the ongoing assembly like a conveyor and a supply AGV that brings parts on request. There are 33 activities in the model, though each can be activated from several different sets of conditions, in total broken down as follows;

- 4 activities are for the AGV and supply AGV, these only allow for requesting and dismissing them.

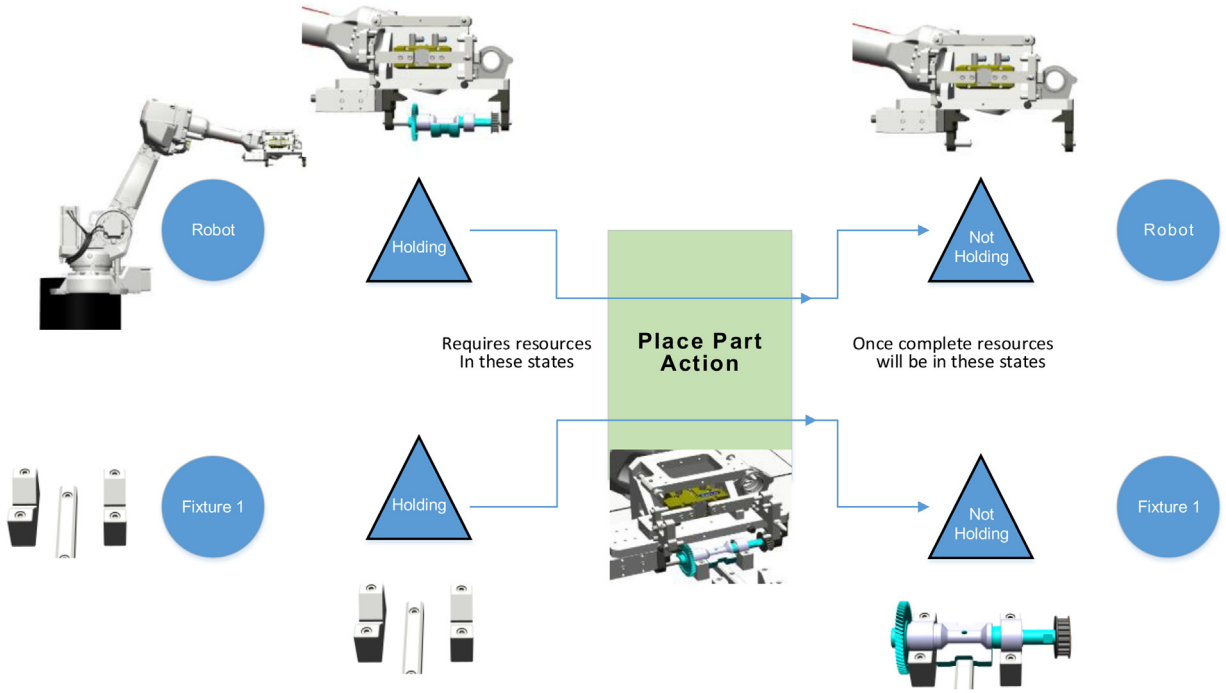

Fig. 4. An example of an action impacting two components of the cell modelled in Fig. 3.

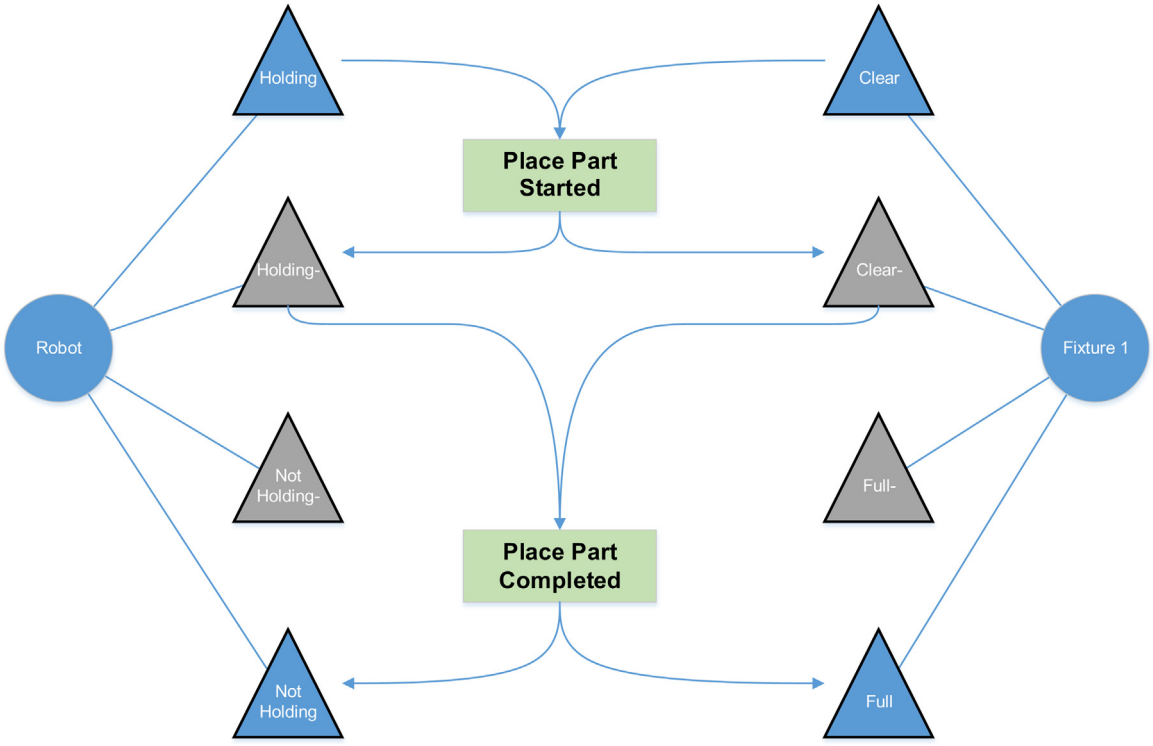

Fig. 5. The transformation of the **action** seen in Fig. 4 giving rise to a pair of events and related states for the resources of the environment. Note that **Holding-** state is created by the described process and could be equivalent to the previously suggested **releasing** state.

- 8 activities are for the robot arm moving between locations in the cell, certain operations can only be performed in certain locations.
- 6 activities are for the robot arm, related to taking and placing parts and performing some operations.
- 14 activities for the workers.
- 1 for a joint worker/robot activity where two workers and the robot carry a large part from the supply AGV to the assembly AGV together.

At any given time there is only a single assembly process underway and when an assembly completes it is possible to continue by

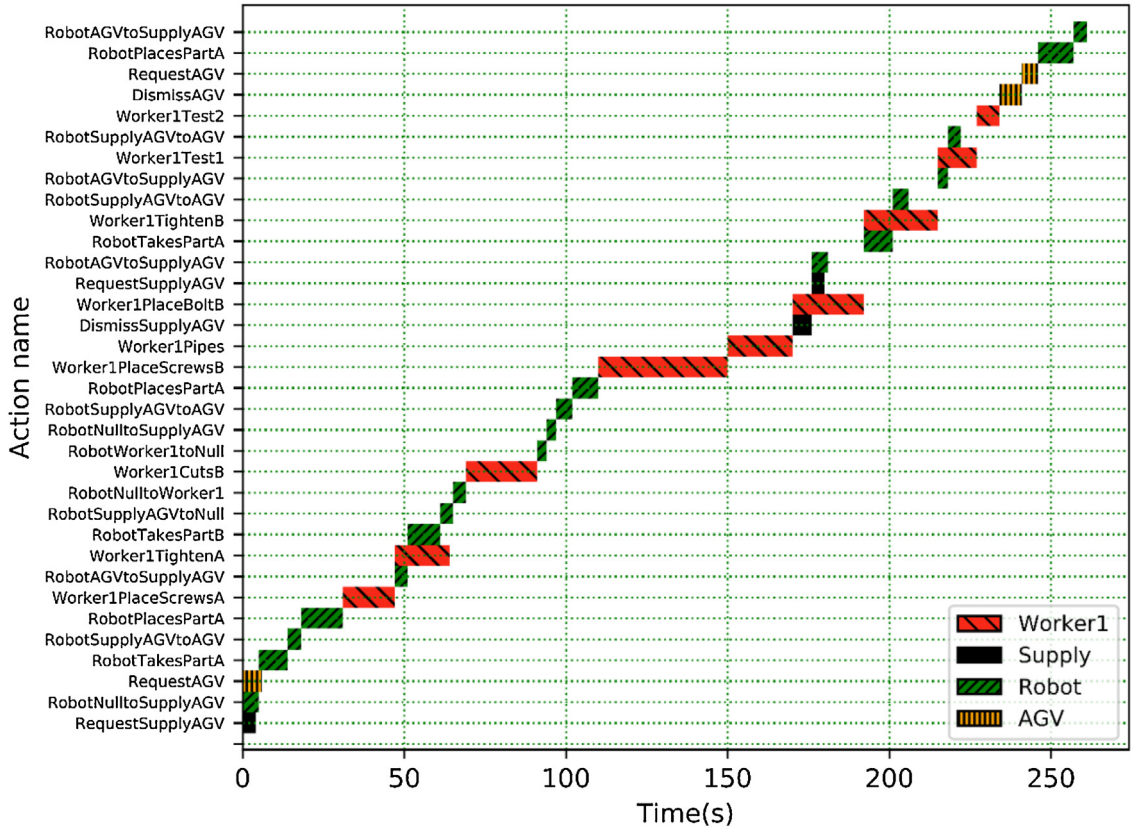

Fig. 6. A chart showing a series of decisions for the simulated case study model with only one human worker active. The cell also includes a supply AGV, a primary AGV and a robot arm assisting the worker.

requesting a new AGV and supply AGV, starting a further assembly.

Fig. 6 depicts the simulation results of the case study when the robot and a single worker are used to perform the task.

The model includes two workers however the second worker is *deactivated*, that is several of the states related to them are set to values that do not fulfil any activity preconditions. This means that the system is unable to find valid start actions for the second worker. The chart shows the interleaving of activities for the two resources and even some collaboration, with the robot holding parts for the human partner to work upon.

Fig. 7 shows an alternative run where both workers are available. More activities happen in parallel and the overall runtime is shorter, as might be expected. The robot still interacts with one or other of the workers where appropriate, or performs other operations while the two workers are otherwise engaged.

3.2. A demonstration assembly station

This example is based on a physical demonstration cell. The cell was created with the help of an automotive manufacture and uses the same robot, conveyor technology and product as one of their real lines. Fig. 8 shows both a diagram and a photograph of the demonstration cell. It consists of a conveyor with a single stopping location, two tables that can each have an ongoing assembly process if a worker is active at the table and a robot that must decide how to share its time between the workers. The conveyor transfers pallets each of which carries two parts, a top and bottom for one assembly process. There is then a final table that acts as a magazine for additional parts used in the assembly.

The model consisted of 26 variables (leaves of the model tree). Activities are divided between the workers and the robot, with one instance of collaboration where the robot can hold a part for inspection by a worker, thus reducing physical strain on the worker. The activities can be broadly classified into two types, those that directly impact an assembly process, and those that change the state of parts of the cell, for example moving parts from the conveyor to an assembly station. Only the relevant set of possible activities have been implemented at this time. The following are the set of 32 activities made available;

- 6 robot movements between areas
- 6 robot activities to take or place parts (top and bottom of the assembly and the completed product)
- 2 robot activities for holding the top up for inspection and then placing it onto the table
- 18 human activities (there are two workers on different tables with 9 activities each)

This demonstration execution begins with a single worker and robot, and hence only a single active assembly. Halfway through, the second table and worker are *activated* resulting in the robot beginning to include the second worker in its activities. A report on this execution can be seen in Fig. 9.

4. Quality of decisions

An evaluation of the method proposed has been conducted in simulation on both the examples presented. The varying team sizes example was evaluated in comparison to two alternative decision making approaches, a purely uniform random method and a method using a greedy heuristic to limit the choices to those that look promising in the immediate future. For both of these approaches, the average time taken for execution was higher than using the MCTS approach, and the full results can be seen in Table 1.

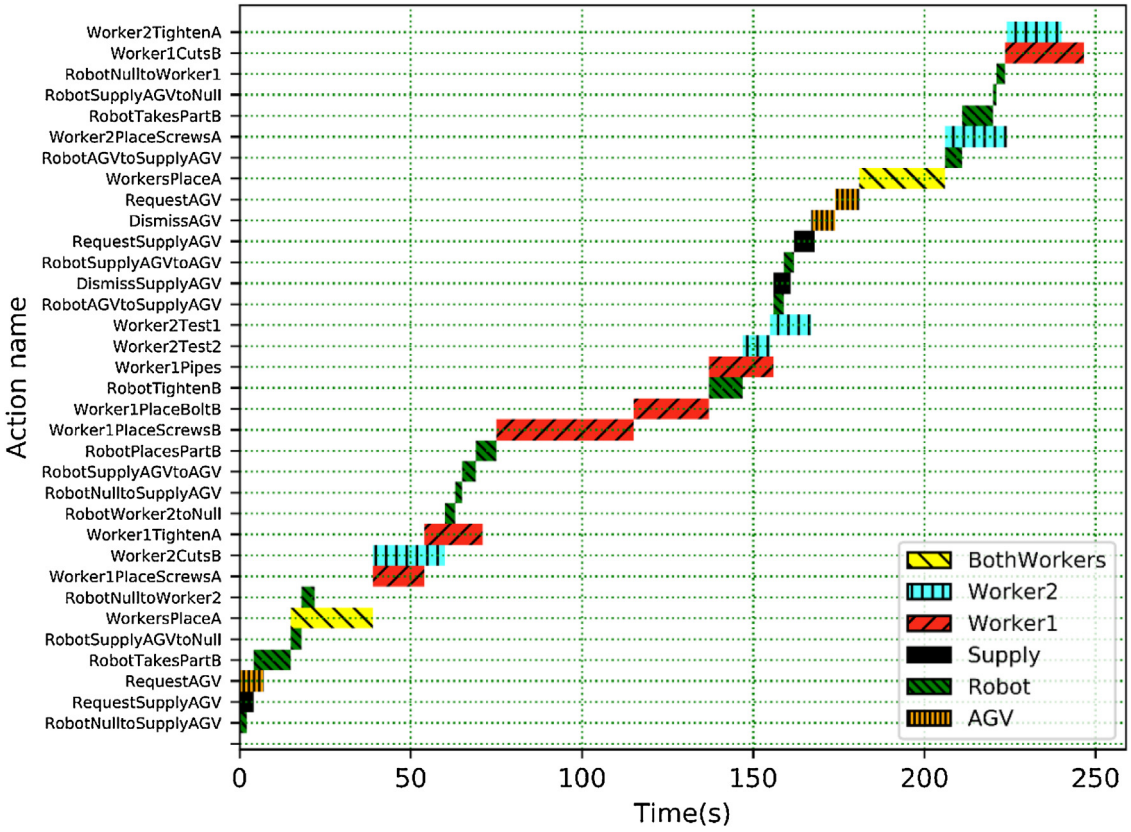

Fig. 7. A chart showing a series of decisions for the same model as Fig. 6 with two workers active.

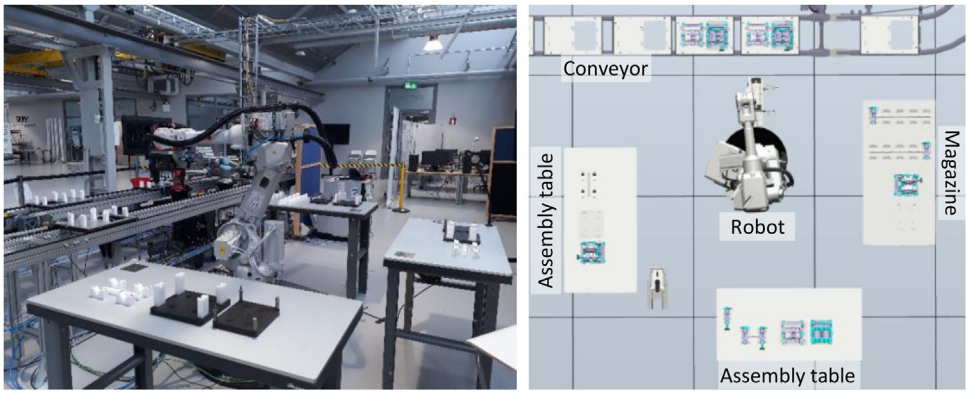

Fig. 8. A photograph and 3D model of the demonstration environment.

We would note that our more modular approach to some of the decisions, such as the robot arm movement between areas of the cell, makes the problem more complex than it might be for a pure scheduler. However, this also reduces the effort needed by the programmer, who need only give the system the possible actions, and allow the dynamic scheduler to manage how they interact, adapting to changing conditions as necessary.

For the example using the physical cell we can compare to (Wang et al., 2019), which sought to examine the multi-objective task of both the ergonomic load on the workers and cycle time. In (Wang et al., 2019) 1 million solutions to the task assignment problem were randomly generated and in respect to cycle time, the best result of 180 feasible solutions had a cycle time of about 290 s. The comparison was done by creating a simulation using our robot path segments, rather than (Wang et al., 2019)'s larger actions and the time information given in their paper for the human worker's actions. The cost of their larger action blocks was checked by adding together the smaller path segments required by our version. The average time taken over 30 simulation runs using the MCTS method was 277.62 s. The cause of this slight improvement appears to be that the smaller operations used in the MCTS system allow for a finer-grained overlap of tasks between workers and robot. The authors of (Wang et al., 2019) noted to us that they did not attempt to optimise the solution but instead focused on characterising the multi-objective optimisation front of the cycle time against the human ergonomic cost. Our result focuses on the time optimisation and shows that the MCTS method does provide an optimising effect on this small example problem.

In the future, we would like to continue to evaluate more traditional methods of scheduling and optimisation for static planning against the use of the MCTS dynamic decision-maker.

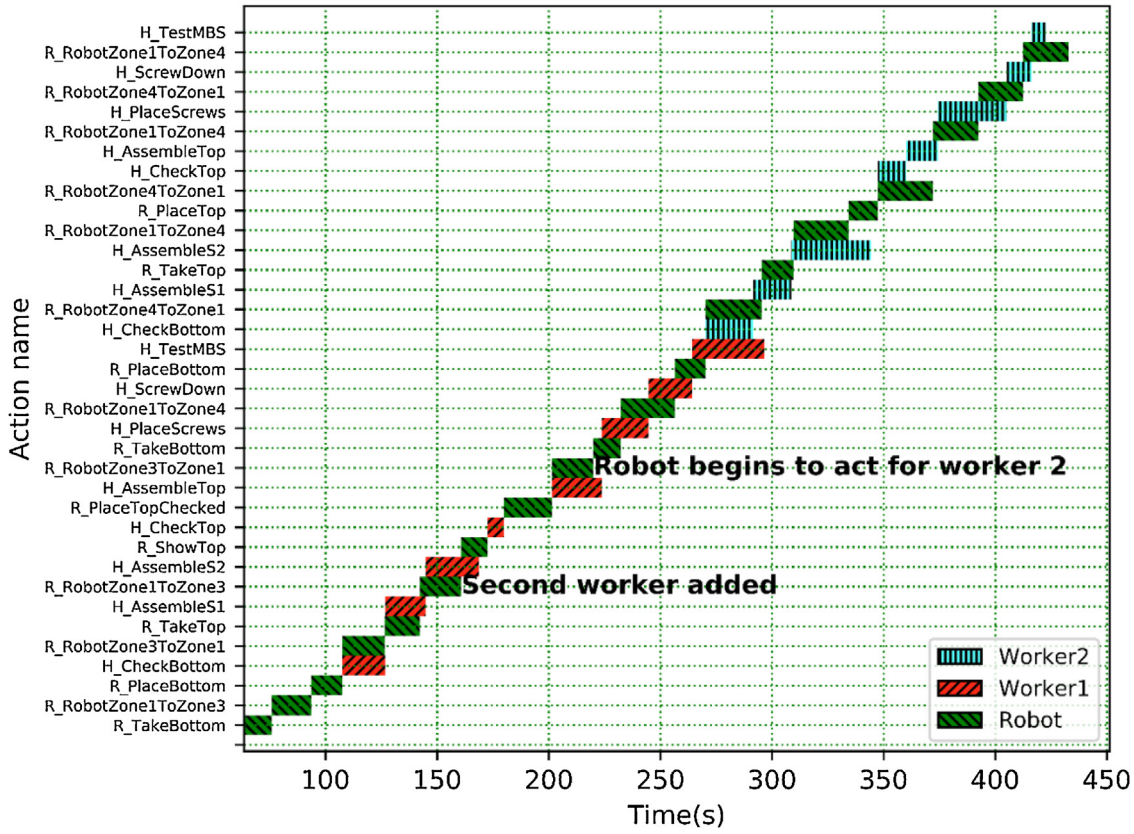

Fig. 9. A report chart of a run using a real robot and two workers.

Table 1
A comparison of MCTS search with purely random and heuristically guided decision strategies over 30 sample executions. Averages are computed on those runs that completed in less than 5 min and the samples that failed to complete in this time are illustrated through the report of the mean or maximum being greater than some value, where appropriate..

Strategy	>5min	Time to completion of a single production cycle [s]		
		Minimum	Mean	Maximum
MCTS	0	177	190.95	218
Uniform Random	14	220	>265.78	>300
Greedy Random	2	200	>244.84	>300

5. Conclusion & future work

Decision making in response to quickly changing conditions is likely to play a greater role in manufacturing as industrial technology continues to move towards greater flexibility in their tools and processes. MCTS has been successfully applied as an AI algorithm in other fields of computing and this paper has described one way in which it can be adapted for use in general factory automation. The approach taken has emphasised the simulation phase of the algorithm, with some short-circuiting of the search in the initial development of the tree. To enable ease of use a model of the production cells in terms of discrete variables and actions guarded by the possible values of these variables has been implemented, an approach also seen in (Belzner, 2017). This model is pre-processed to give rise to a more verbose set of possible values, both for the variables and actions which initialise or represent the completion of activities in the cell. The more verbose model is then used by the decision making process itself as seen in (Toussaint et al., 2016). The tool created has been demonstrated through example applications and performance evaluations against an offline analysis of the real example cell and a heuristic search method. The performance evaluations suggest that the approach taken is at least as effective as these alternatives while we would argue it is more flexible for usage in a real production environment, for example in terms of adding new operations to an existing cell.

It was pointed out in the introduction that other researchers have found that MCTS controls the combinatorial explosion experienced in many optimisation and decision problems. Despite the advantages presented by MCTS, there is a relationship between the size of a problem and how well the algorithm can achieve sensible estimates of the value of decisions. While the approach presented here seems to work quite well it will be of interest to apply the approach on larger examples. In the state-based approach employed here, more variables modelling the state of an environment, more activities and increasing the number of possible simultaneous activities will increase the size of the problem, so a larger problem will require more simulation runs than a smaller problem to achieve the same level of data about all expected future options. It was also pointed out that other researchers have examined the optimality gap between a solution from MCTS and a solution from an exact algorithm, but such a study of the performance of this approach has not yet been done and will be required in the future.

The method used here has, as explained, modified the original approach of MCTS making it more like a repeated DES. Successful MCTS implementations increase the detail of the tree previously computed with time, preserving that knowledge between decisions, improving both the performance of the algorithm and the quality of decisions made. This is managed through tree *policies* for the selection and expansion of nodes in the tree. It would be of great interest to return to this concept and try to adapt the algorithm again to add and experiment with various policies while applying it to the multi-device environment with simultaneous activities which is used here. Other researchers have used neural networks, trained against offline data, to learn these policies (Silver et al., 2016). This approach would be in line with the concept of big-data for factory environments. The approach would be to gather performance data on activities performed, perform offline training of the neural network to derive policies for MCTS within the simulated environment and then downloading the neural networks to the devices in the factory. Safety of the decisions would still be enforced, such as it can be, through the state-action model and the MCTS search, but optimality should be improved. In conclusion, this paper presents a method for adaptation of MCTS to factories which both appears to work well and has the possibility for a range of improvements that would help to realise the full promise of the smart factory.

Finally we have noted in several places some design decisions and limitations that should be further investigated in the future including; updating of the time in an existing search tree rather than restarting the search, treating the expected time of activities as a probability function rather than as a fixed value and return to a more standard use of expansion, selection and simulation.

Author statement

Richard Senington: Conceptualization, Methodology, Software, Writing – Original draft preparation.

Bernard Schmidt: Visualization, Writing - Review & Editing.

Anna Syberfeldt: Project administration, Funding acquisition, Writing - Review & Editing.

Declaration of Competing Interest

The authors report no declarations of interest.

Acknowledgements

This paper is based on work performed within the research projects TWIN [20160297] sponsored by the Swedish Knowledge Foundation and SYMBIO-TIC [637107] sponsored by the European Union.

References

Abramson, B.D., 1987]. The Expected-outcome Model of Two-player Games. Columbia University, New York, NY, USA.

Belzner, L., 2017]. Monte Carlo action programming. ArXiv, online.

Bouzy, B., Helmstetter, B., 2004]. Monte-Carlo go developments. In: Advances in Computer Games: Many Games, Many Challenges. Springer US, Boston, MA, USA, pp. 159–174.

Brown, N.G., Tellex, S., 2015]. Modeling and Solving Human-robot Collaborative Tasks Using POMDPs.

Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S., 2012]. A survey of Monte Carlo Tree Search methods. IEEE Trans. Comput. Intell. AI Games 4 (1), 1–43.

Chaslot, G.M.J.B., Jong, Sd., Saito, J.T., Uiterwijk, J.W.H.M., 2006]. Monte-Carlo tree search in production management problems. BNAIC'06: Proceedings of the 18th Belgium-Netherlands Conference on Artificial Intelligence.

Chaslot, G., Bakkes, S., Szita, I., Spronck, P., 2008a]. Monte-Carlo Tree Search: a New framework for game AI. In: Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment, Stanford, California, USA.

Chaslot, G., Winands, M., Herik, H., Uiterwijk, J., Bouzy, B., 2008b]. Progressive strategies for Monte-Carlo Tree Search. New Math. Nat. Comput. 4, 343–357.

Coulom, R., 2006]. Efficient selectivity and backup operators in Monte-Carlo Tree Search. In: 5th International Conference on Computer and Games, Turin, Italy.

Dahl, M., Bengtsson, K., Bergagård, P., Fabian, M., Falkman, P., 2017]. Sequence planner: supporting integrated virtual preparation and commissioning. IFAC-PapersOnLine 50.

Edelkamp, S., Gath, M., Greulich, C., Humann, M., Herzog, O., Lawo, M., 2016]. Monte-Carlo Tree Search for logistics. In: Commercial Transport. Lecture Notes in Logistics. Springer, Cham, pp. 427–440.

Kardos, C., Kemény, Z., Kovács, A., Pataki, B.E., Váncza, J., 2018. Context-dependent multimodal communication in human-robot collaboration. Procedia CIRP 72, 15–20.

Kartal, B., Nunes, E., Godoy, J., Gini, M., 2016]. Monte Carlo Tree Search with branch and bound for multi-robot task allocation. The IJCAI'16 Workshop on Autonomous Mobile Service Robots.

Keller, T., Helmert, M., 2013]. Trial-based heuristic tree search for finite horizon MDPs. In: Proceedings of the Twenty-Third International Conference on International Conference on Automated Planning and Scheduling, Rome, Italy.

Koren, Y., 2010]. Reconfigurable manufacturing systems. In: The Global Manufacturing Revolution. John Wiley & Sons, Ltd, pp. 227–252.

Koren, Y., Gu, X., Guo, W., 2018]. Reconfigurable manufacturing systems: principles, design, and future trends. Front. Mech. Eng. 13 (2), 121–136.

Krüger, J., Wang, L., Verl, A., Bauernhansl, T., Carpanzano, E., Makris, S., Fleischer, J., Reinhart, G., Franke, J., Pellegrinelli, S., 2017]. Innovative control of assembly systems and lines. CIRP Ann. Manuf. Technol. 66 (2), 707–730.

Liebana, D.P., Rohlfshagen, P., Lucas, S., 2012. Monte Carlo Tree Search: Long-term versus short-term planning. In: IEEE Conference on Computational Intelligence and Games (CIG), Granada, Spain.

Lubosch, M., Kunath, M., Winkler, H., 2018]. Industrial scheduling with Monte Carlo Tree Search and machine learning. Procedia CIRP 72, 1283–1287.

Morariu, C., Morariu, O., Raileanu, S., Borangiu, T., 2020]. Machine learning for predictive scheduling and resource allocation in large scale manufacturing systems. Comput. Ind. 120, 103244.

Nikolakis, N., Kousi, N., Michalos, G., Makris, S., 2018. Dynamic scheduling of shared human-robot manufacturing operations. Procedia CIRP 72, 9–14.

Ouelhadj, D., Petrovic, S., 2008]. A survey of dynamic scheduling in manufacturing systems. J. Sched. 12 (4), 417–431.

Samothrakis, S., Roberts, S., Perez Liebana, D., Lucas, S., 2014. Rolling horizon methods for games with continuous states and actions. In: IEEE Conference on Computational Intelligence and Games (CIG), Dortmund, Germany.

Sandrin, E., Trentin, A., Forza, C., 2014]. Organizing for mass customization: literature review and research agenda. Bonfring Int. J. Ind. Eng. Manage. Sci. 5, 159–167.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D., 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529 (7587), 484–489.

Tang, D., Dai, M., Salido, M.A., Giret, A., 2016]. Energy-efficient dynamic scheduling for a flexible flow shop using an improved particle swarm optimization. Comput. Ind. 81, 82–95.

Telgen, D.H., 2017]. Grid Manufacturing : A Cyber-physical Approach With Autonomous Products and Reconfigurable Manufacturing Machines.

Toussaint, M., Munzer, T., Mollard, Y., Yang Wu, L., Anh Vien, N., Lopes, M., 2016. Relational activity processes for modeling concurrent cooperation. In: ICRA) IEEE International Conference on Robotics and Automation, Stockholm, Sweden.

Trunda, O., Barták, R., 2013]. Using Monte Carlo Tree Search to solve planning problems in transportation domains. In: Advances in Soft Computing and Its Applications. MICAI, Heidelberg.

Valente, A., De Simone, L., Carpanzano, E., Ferraris, F., 2012]. A methodology for static and dynamic scheduling of automation tasks in reconfigurable production systems. CIRP J. Manuf. Sci. Technol. 5 (4), 241–253.

Wang, W., Bandaru, S., Sánchez de Ocãna Torroba, A., 2019]. Improved human-robot collaboration through simulation-based optimization. In: Advances in Manufacturing Technology XXXIII : Proceedings of the 17th International Conference on Manufacturing Research, Belfast, UK.

Williams, P., Walton-Rivers, J., Perez Liebana, D., Lucas, S., 2015]. Monte Carlo Tree Search applied to co-operative problems. In: IEEE Conference on Computer Science and Electronic Engineering (CEEC), Colchester, UK.

Glossary

AGV: Automated Guided Vehicle
DES: Discrete Event Simulation
HRC: Human-Robot Collaboration
MCAP: Monte Carlo Action Programming
MCTS: Monte Carlo Tree Search